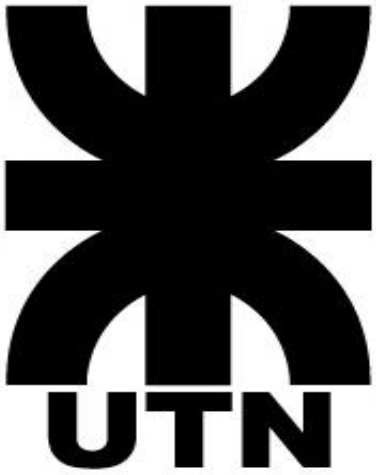

Universidad Tecnológica Nacional
Facultad Regional Concepción del Uruguay

Ingeniería en Sistemas de Información

Sistemas Operativos

Trabajo Práctico Final:
"Ampliación y mejora del shell experimental"

Integrantes del Grupo:

Arrúa, Martin
Lazbal, David
Rivera, Ramiro

Fecha de Entrega:
19/02/2015

Docentes:

Ing. Arellano Gabriel
Ing. Aguiar Osvaldo

```
UNIT utilidades;

INTERFACE

      USES
            strings, baseunix, sysutils, dateutils;

      Const
            N = 300;                // Número máximo de archivos a listar en un
directorio (ls)
            P = 3;                        // Número máximo de parámetros para
los comandos (sin contar directorios)
            LF = #10;               // Caracter Line Feed.
            CR = #13;               // Caracter Carriage Return.
            CRLF = CR + LF;         // CRLF

      Type
            vDirent = array [1..N] of Dirent;              // Vector para alma-
cenar nombres de archivos de un directorio y así ordenarlos.
            parametros= array [1..P] of AnsiString;        // Vector para alma-
cenar parámetros de comandos.

      var
            stdOutPut: Text;              // Archivo de redirección de la salida
estándar.
            strOutPut: string;                  // Nombre del archivo de redirec-
ción.
            newInPut : Text;              // Redirección de la entrada estándar
para comando pipe.
            flagInPut: Boolean;                 // Se le asigna momentaneamente
true durante un pipe.


//  UTILIDADES
function puntos(s: string): integer;
            // Cantidad de puntos '.' en un string.
function dirActual: string;
            // Devuelve la dirección de trabajo actual.
function dirHome: string;
                  // Devuelve el directorio Home. Por defecto /home/user.
function strSin(param1, param2: string): string;
      // elimina param2 de la cadena param1.
procedure burbujaDirent(var v: vDirent; max: integer);
      // ordena los archivos listados por ls. Algoritmo burbuja.
function redireccion(s: string): boolean;
      // Devuelve true si encuentra los operadores '>' o '>>' en la cadena.
function tuberia(s: string): boolean;
            // Devuelve true si encuentra el operador '|' en una cadena.
function enBg(s: string): boolean;
            // Devuelve true si el proceso debe ejecutarse en background.
function solicitudBG(var s: string):boolean;
      // Verifica si se ingreso "bg", la solicitud para ejecutar un trabajo en
segundo plano.
function solicitudFG(var s: string):boolean;
      // Verifica si se ingreso "fg", la solicitud para ejecutar un trabajo en
primer plano.
procedure Mostrarerror(n: integer);
            // Devuelve el mensaje de error <n>.
```

```
function asString(V:int64):string;
            // Convierte un valor a string.
function tiempoUnixAHumano (tUnix:int64):string;
      // Convierte el valor de tiempo que devuelve Unix.
function permisosACadena(path:string):string;
      // Verifica si el usuario tiene permisos de lectura, escritura y ejecu-
ción, y lo imprime.
function tamAsString(tam:int64; espacios: byte):string;
      // Permite alinear un string.
function extraerComando(var s:string):string;
      // Devuelve el comando de la cadena (o primer componente antes del primer
espacio), y se elimina el componente de la entrada recibida.
function extraerArgumentos(var s:string;var i:integer):parametros;          //
Devuelve un vector de argumentos de la forma -* y se eliminan los argumentos de
la entrada recibida.
function modoACadena(modo:integer):string;
            // Traduce los permisos de un archivo
function tipoArchivoAF(modo:integer):string;
      // Traduce los tipos de archivos para el comando LS con parámetros A y/o F
function tipoArchivoL(modo:integer):string;
            // Traduce los tipos de archivos para el comando LS con parámetro L
function nombreComandoDesdeRuta(s:string):string;
      // Dada una dirección absoluta a un comando, devuelve sólo el nombre del
mismo.
function paramValido(str:string):boolean;
      // Determina si un parámetro del ls es válido o no.
function stringToAnsiString(str:string):ansiString;
      // Convierte un string en un ansiString
function ansiStringToString(str: ansiString):string;                          //
Convierte un ansiString en un string
function sinEspacios(s:string):string;
            // Elimina los espacios existentes en una cadena
function eliminarSubcadena(cadena,subcadena:string):string;                    //
Elimina una subcadena de la cadena original

IMPLEMENTATION

function eliminarSubcadena(cadena,subcadena:string):string;
var auxStr:string;
      posi:longint;
begin
      auxStr:=cadena;
      posi:=0;
      posi:=pos(upcase(subcadena),upcase(cadena));
      if posi<>0 then
            delete(auxStr,posi,length(subcadena));
      eliminarSubcadena:=auxstr;
end;

function sinEspacios(s:string):string;
var strAux:string; I:longint;
begin
      straux:='';
      for I:=1 to length(s) do
            begin
            if s[I]<>#32 then
                  strAux:=strAux+s[I];
            end;
      sinEspacios:=strAux;
```

```
end;


function asString(V:int64):string;
var S:string;
begin
      STR(V,S);
      asString:=S;
end;


function puntos(s: string): integer;
var I: integer;
Begin
      puntos:= 0;
      for I:= 1 to length(s) do
            if s[I] = #46 then puntos:= puntos + 1;
End;


function dirActual: string;
Begin
      getDir(0,dirActual);
End;


function dirHome: string;
Begin
      dirHome:= fpGetEnv('HOME');
End;


function strSin(param1, param2: string): string;
Begin
      strSin:= rightStr(param1,((length(param1))-(length(param2))));
End;


procedure burbujaDirent(var v: vDirent; max: integer);
var I,J: integer;
      aux: Dirent;
Begin
      for I:= 1 to max-1 do
       for J:= 1 to max-I do
        if upcase(strPas(v[J].d_name)) > upcase(strPas(v[J+1].d_name)) then
            Begin
                  aux := v[J+1];
                  v[J+1]:= v[J];
                  v[J] := aux;
            End;
End;


function redireccion(s: string): boolean;
Begin
      redireccion:= (pos(' > ',s) <> 0) or (pos(' >> ',s) <> 0);
End;
```

```
function tuberia(s: string): boolean;
Begin
      tuberia:= (pos(' | ',s) <> 0);
End;


function enBg(s: string): boolean;
Begin
      enBg:= ((length(s) <> 1) and (pos(' &',s) = length(s)-1)); // Debe ser lo
ultimo escrito
End;


function solicitudBG(var s: string):boolean;
Begin
      solicitudBG:= 'BG'= upcase(s);
End;


function solicitudFG(var s: string):boolean;
begin
      solicitudFG:='FG'=upcase(s);
end;


procedure Mostrarerror(n: integer);
var
      errorAux:string;
Begin
      case n of
      1:          errorAux:= 'Error n° 1: Parámetros insuficientes.';
      2:          errorAux:= 'Error n° 2: No se puede reconocer el comando.';
      3:          errorAux:= 'Error n° 3: Falla en el Fork, no es posible ejecu-
tar el nuevo proceso.';
      4:          errorAux:= 'Error n° 4: El directorio no existe, o no puede
ser accedido.';
      5:          errorAux:= 'Error n° 5: Comando no reconocido.';
      6:          errorAux:= 'Error n° 6: Error en Exec, no es posible ejecutar
el proceso.';
      7:          errorAux:= 'Error n° 7: El programa no se encuentra en el
Path.';
      8:          errorAux:= 'Error n° 8: Parámetros incorrectos.';
      9:          errorAux:= 'Error n° 9: Los parámetros deben ser de tipo numé-
rico.';
      10:   errorAux:= 'Error n° 10: Error al intentar abrir el/los ar-
chivo(s).';
      11:   errorAux:= 'Error n° 11: Error de redirección de la salida estándar
- Operador ''>''';
      12:   errorAux:= 'Error n° 12: Error de redirección de la salida estándar
- Operador ''>>''';
      13:   errorAux:= 'Error n° 13: Error al intentar crear tuberías.';
      20:         errorAux:= 'Error n° 20: Trabajo no encontrado.';
      else  errorAux:= 'Error desconocido.';
      end;
      writeln(errorAux);
      if n in [1,2,5,8] then
            writeln('Si necesita ayuda utilice el comando "help".');
      exit;
End;
```

```
function numAMes(numMes:word):string;
var
	Aux:string;
begin
	case numMes of
		1:	Aux:='ene';
		2:	Aux:='feb';
		3:	Aux:='mar';
		4:	Aux:='abr';
		5:	Aux:='may';
		6:	Aux:='jun';
		7:	Aux:='jul';
		8:	Aux:='ago';
		9:	Aux:='sep';
		10:	Aux:='oct';
		11:	Aux:='nov';
		12:	Aux:='dic';
		end;
numAMes:=aux;
end;

function tiempoUnixAHumano (tUnix:int64):string;
var
	Y,Mo,D,H,Mi,S,MS : Word;
	Aux:string;
begin
	DecodeDateTime(UnixToDateTime(tUnix),Y,Mo,D,H,Mi,S,MS);
	Aux:=numAMes(Mo)+#32;
	if D<10 then
		Aux:=Aux+#32+asString(D)+#32
	else
		Aux:=Aux+asString(D)+#32;
	if Y<2014 then
		Aux:=Aux+#32+asString(Y)+#32
	else
		begin
		if H<10 then
			Aux:=Aux+'0'+asString(H)+':'
		else
			Aux:=Aux+asString(H)+':';
		if Mi<10 then
			Aux:=Aux+'0'+asString(Mi)+#32
		else
			Aux:=Aux+asString(Mi)+#32;
		end;
	tiempoUnixAHumano:=Aux;
end;

function permisosACadena(path:string):string;
var
	Aux:String;
begin
	Aux:='---';
	if fpAccess (path,R_OK)=0 then Aux[1]:='r';
	if fpAccess (path,W_OK)=0 then Aux[2]:='w';
	if fpAccess (path,X_OK)=0 then Aux[3]:='x';
	permisosACadena:=Aux+#32;
end;
```

```
function tamAsString(tam:int64; espacios: byte):string;
var
      aux,buff:string;
      i:integer;
begin
      buff:='';
      aux:=asString(tam);
      for i:=1 to espacios-length(aux) do
            buff:=buff+' ';
      tamAsString:=buff+Aux+#32;
end;

function extraerComando(var s:string):string;
var aux:string;e:integer;
begin
      while (s<>'') and (s[1]=' ') do
            delete(s,1,1);                              //eliminar espacios ini-
ciales
      if (s<>'') then
      begin
            e:=1;
            aux:='';
            repeat                                                  //comienza detección
de primer componente
            begin
                  aux:=aux + s[e];                          //agrega un caracter
                  if s[e] = '"' then                              //si el ultimo
elemento agregado es ".
                  begin
                        inc(e);
                        while s[e] <> '"' do                //agrega todo hasta
el proximo ".
                        begin
                              aux:=aux + s[e];
                              inc(e);
                        end;
                        aux:=aux + s[e];                    //agrega las " de
cierre
                  end;
                  inc(e);
            end;
            until (e>length(s))OR(s[e] = ' ');                        //se en-
contró un espacio fuera de ""
            extraerComando:= aux;                   //devolver comando
            s:=copy(s,length(aux)+1,length(s)); //la entrada se recorta hasta el
primer parametro
            while (s<>'') and (s[1]=' ') do
                  delete(s,1,1);                            //eliminar espacios
iniciales
      end
      else
            extraerComando:=s;
end;

function extraerArgumentos(var s:string;var i:integer):parametros;
var j:integer;param:ansiString;
begin
      for j:=1 to P do
```

```
            extraerArgumentos[j]:='';           //inicializar vector de argumen-
tos
      while (s<>'')AND(s[1]=' ') do
            delete(s,1,1);                              //eliminar espacios inicia-
les
      i:=0;
      param:='';
      while (i<=N)AND(s<>'')AND(s[1]='-') DO          //comienza detección de
primer componente
      begin
            delete(s,1,1);          //elimina el guion que indica parametro
            WHILE (s<>'')AND(s[1]<>' ') DO
            begin
                  param:=param + s[1];
                  delete(s,1,1);
            end;
            inc(i);
            extraerArgumentos[i]:=param;
            param:='';
            WHILE (s<>'')AND(s[1]=' ') DO
                  delete(s,1,1);                        //eliminar espacios
entre parametros
      end;
end;

function modoACadena(modo:integer):string;
var
      Aux:String;
begin
      Aux:='----------';
      if (modo and S_IFDIR) = S_IFDIR then Aux[1]:='d';
      if (modo and S_IRUSR) = S_IRUSR then Aux[2]:='r';
      if (modo and S_IWUSR) = S_IWUSR then Aux[3]:='w';
      if (modo and S_IXUSR) = S_IXUSR then Aux[4]:='x';
      if (modo and S_IRGRP) = S_IRGRP then Aux[5]:='r';
      if (modo and S_IWGRP) = S_IWGRP then Aux[6]:='w';
      if (modo and S_IXGRP) = S_IXGRP then Aux[7]:='x';
      if (modo and S_IROTH) = S_IROTH then Aux[8]:='r';
      if (modo and S_IWOTH) = S_IWOTH then Aux[9]:='w';
      if (modo and S_IXOTH) = S_IXOTH then Aux[10]:='x';
      modoACadena:=Aux+#32;
end;

function tipoArchivoAF(modo:integer):string;
begin
      if fpS_ISLNK(modo) then
            tipoArchivoAF:=('(L)')
            else
                  if fpS_ISCHR(modo) then
                        tipoArchivoAF:=('(C)')
                  else
                        if fpS_ISREG(modo) then
                              tipoArchivoAF:=('(F)')
                        else
                              if fpS_ISDIR(modo) then
                                    tipoArchivoAF:=('(D)')
                                    else
                                          if fpS_ISBLK(modo) then
                                                tipoArchivoAF:=('(B)')
```

```
                                        else
                                              if fpS_ISFIFO(modo) then
                                                    tipoArchivoAF:=('(I)')
                                              else
                                                    if fpS_ISSOCK(modo) then
                                                          tipoAr-
chivoAF:=('(S)');
end;


function tipoArchivoL(modo:integer):string;
begin
      if fpS_ISLNK(modo) then
            tipoArchivoL:=('LNK'+#32)
            else
                  if fpS_ISCHR(modo) then
                        tipoArchivoL:=('CHR'+#32)
                  else
                        if fpS_ISREG(modo) then
                              tipoArchivoL:=('FILE')
                        else
                              if fpS_ISDIR(modo) then
                                    tipoArchivoL:=('DIR'+#32)
                              else
                                    if fpS_ISBLK(modo) then
                                          tipoArchivoL:=('BLK'+#32)
                                    else
                                          if fpS_ISFIFO(modo) then
                                                tipoArchivoL:=('FIFO')
                                          else
                                                if fpS_ISSOCK(modo) then
                                                      tipoArchivoL:=('SOCK');

end;

function nombreComandoDesdeRuta(s:string):string;
var aux:string;
begin
      aux:= s;
      while (pos('/',aux) <> 0) do
            aux:=copy(aux,pos('/',aux)+1,length(aux));
      nombreComandoDesdeRuta:=aux;
end;

function paramValido(str:string):boolean;
begin
      str:=UPCASE(str);
      paramValido:=(
                              (str='A') or (str='F') or (str='L') or
                              (str='LA') or (str='LF') or (str='AL') or
(str='AF') or (str='FL') or (str='FA') or
                              (str='LFA') or (str='LAF') or (str='FLA') or
(str='FAL') or (str='AFL') or (str='ALF')
                        )
end;

function stringToAnsiString(str:string):ansiString;
begin
      stringToAnsiString:=str;
```

```
end;

function ansiStringToString(str: ansiString):string;
begin
      ansiStringToString:=str;
end;

BEGIN
      {
            No utilizado, se guarda por compatibilidad con OShell
      }
      strOutPut:= '';                           // StrOutPut refiere a la redi-
rección de la salida estándar.
      setTextLineEnding(stdOut,CRLF); // Asignándole formato a la salida están-
dar.
      flagInPut:= false;
END.
```